Steps Toward Optimal Competitive Scheduling

Jeremy Frank and **James Crawford** and **Lina Khatib** * and **Ronen Brafman** †
Computational Sciences Division
NASA Ames Research Center, MS 269-3
frank@email.arc.nasa.gov
Moffett Field, CA 94035

Introduction

This paper is concerned with the problem of allocating a unit capacity resource to multiple users within a pre-defined time period. The resource is indivisible, so that at most one user can use it at each time instance. However, different users may use it at different times. The users have *independent, selfish* preferences for *when* and for *how long* they are allocated this resource. Thus, they value different resource access durations differently, and they value different time slots differently. We seek an optimal allocation schedule for this resource.

This problem arises in many institutional settings where, e.g., different departments, agencies, or personal, compete for a single resource. We are particulary motivated by the problem of scheduling NASA's Deep Space Satelite Network (DSN) among different users within NASA. Access to DSN is needed for transmitting data from various space missions to Earth. Each mission has different needs for DSN time, depending on satelite and planetary orbits. Typically, the DSN is over-subscribed, in that not all missions will be allocated as much time as they want. This leads to various inefficiencies – missions spend much time and resource lobbying for their time, often exagerating their needs. NASA, on the other hand, would like to make optimal use of this resource, ensuring that the good for NASA is maximized. This raises the thorny problem of how to measure the utility to NASA of each allocation.

In the typical case, it is difficult for the central agency, NASA in our case, to assess the value of each interval to each user – this is really only known to the users who understand their needs. Thus, our problem is more precisely formulated as follows: find an allocation schedule for the resource that maximizes the sum of users preferences, when the preference values are private information of the users. We bypass this problem by making the assumptions that one can assign money to customers. This assumption is reasonable; a committee is usually in charge of deciding the priority of each mission competing for access to the DSN within a time period while scheduling. Instead, we can assume that the committee assigns a budget to each mission. We then assume that customers express preferences by attaching a monetary value to each allocation of an interval, and that the utility to NASA is linear in the sum of user values.

Given the users' valuations over different time allocations, the problem is a challenging optimization scheduling. However, as we noted, these valuations are private information known only to the users, and *a-priori* we have no reason to assume that they will describe it truthfully – in practice, missions tend to exagerate these values in order to obtain more DSN time. This problem is reminiscent of similar competitive allocation problems. Many such problems are solved using auction-based mechanisms. In our approach we adopt the VCG mechanism for auction clearing and payment determination to give customers (bidders) no incentive to lie. This paper is focusing on the scheduling optimization aspect of our problem and, therefore, we omit further elaboration on the game theory aspect of our solution.

In summary, our problem is to optimally schedule requests over single resource that are, each, specified over a window with flexible duration augmented by preference' values. The problem is over-subcribed and, therefore, some requests will be rejected. To the best of our knowledge, preference elicitation for scheduling with this wide range of features has not been addressed in the literature.

The paper is organized as follows. In Section 2 we review the relevant related background. In Section 3 we formally define the OCS problem. In Section 4 we briefly discssus a VCG-like mechanism for OCS. In Section 5 we describe simplified classes of OCS introducing various relaxations of requirements and their solvers. Preliminary empirical results are presented in Section 6. We conclude in Section 7.

Background

Our problem has two central aspects represent by inducing the agents to reveal their true preferences and then solving the associated optimal scheduling problem. Previous work roughly falls into these two categories. We tackle the first aspect by adopting the Vickery-Clarke-Groves (VCG) mechanism that is incentive compatible in terms of bidding true values (expressing true preferences in our case). We choose to focus on the second aspect in this paper.

The literature on scheduling contains vast numbers of tractable cases for scheduling problems with one optimization criteria. The work most related to our type of optimization problem is that on STPPs. In (?) it was shown that if

*QSS
†QSS

the preference functions over $|e_i - e_j|$ are restricted to be piecewise linear and convex, the STPP can be solved by formulating an appropriate linear program (LP), rendering its solution time polynomial. It is important to note that in an STPP, all events must be assigned a positive time. If we view events as task start and end times, this means that in order to formulate a scheduling problem as an STPP, we must first resolve all resource conflicts either by rejecting certain tasks or by ordering tasks. Thus, in practice the input consists of a set of ordered tasks, all of which must be allocated some time. Cases where there are too many tasks or sets of tasks have trivially infeasible constraints cannot be resolved by the existing techniques.

Definition 1 *An STPP consists of a set of events E, a set of simple temporal constraints of the form $a \leq |e_i - e_j| \leq b$ where $e_i, e_j \in E$, and a set of real-valued preference functions on some event distances: $f(|e_i, e_j|)$. One event, e_0, is designated as the origin (i.e., $e_0 = 0$), and constraints involving e_i and e_0 are interpreted as unary bounds on when e_i can occur. The problem is to find a feasible assignment to $e_i (\neq e_0)$ i.e., an assignment satisfying the constraints maximizing $\sum_{e_i,e_j} f(|e_i - e_j|)$ that also satisfies all constraints.*

The Optimal Competitive Scheduling Problem for the Deep Space Network

We begin with a resource R, a time horizon $[0..h]$, and a set of customers (bidders) J. Let $\mathcal{A}$ be a set of activities (requests). Let S_A be the start time, D_A be the duration, and E_A be the end time of activity $A \in \mathcal{A}$. Associated with A are the following constraints: $S_A + D_A = E_A$, All activities A also share a single unary resource R (i.e. jobs can't overlap). We may also associate other constraints among collections of activities. The most elementary such constraints are absolute start and end time: $0 \leq s_{A_{lb}} \leq S_A \leq s_{A_{ub}} \leq h$, and $0 \leq e_{A_{lb}} \leq E_A \leq e_{A_{ub}} \leq h$. Similarly, tasks can have minimum and maximum durations: $0 \leq d_{A_{lb}} \leq D_A \leq d_{A_{ub}} \leq h$. Another simple form of constraint are Simple Temporal Constraints (**?**) of the form $a \leq T_A - T_B \leq b$, where $T_A \in \{S_A, E_A\}$ and $T_B \in \{S_B, E_B\}$, which can enforce activity orderings and activity endpoint separations.

Customers express preferences over *when* individual activities take place, *how much* activities are separated, and *how long* activities last. For example, a common preference is that longer duration activities are better. Another common preference is for activities to start at some time t. In the DSN setting, the first of these preferences is the most common. Thus, customers' preferences are functions of the form $v_A(S_A, D_A)$.

Briefly, we indicate how the problem is stated as game theory. The auction protocol is a single sealed-bid auction; players/customers bid, winners and prices are determined, and winners determine the schedule that will be executed. Thus, the winning bids must collectively define feasible schedules.

The Optimal Competitive Scheduling Problem (OCS) is to maximize the sum of the value of customers' true preferences over schedules given the constraints and bids by choosing a subset of bids $\mathcal{A}'$ to grant (satisfy) and choosing the values of $S_A, E_A, D_A \forall A \in \mathcal{A}'$ satisfying the relevant constraints.

Problem Complexity

OCS is an extension of $1|r_i; p_i; d_i| \sum_i w_i U_i$ (**?**); that is, scheduling tasks with release times, due dates, and duration constraints on a unary resource in order to minimize the weighted penalty of missed jobs (alternatively maximize value of completed jobs) with duration-dependent values for tasks, additional opportunities for positive value (start time preferences), and additional constraints (simple temporal constraints on timepoints). Karp has shown that $1|r_i; p_i; d_i| \sum_i w_i U_i$ is $\mathcal{NP}$-complete; thus, OCS must be $\mathcal{NP}$-hard. (IS THIS CORRECT JEREMY?)

Tractable Cases: OCS Problem Relaxations

We showed earlier that the OCS problem is $\mathcal{NP}$-hard. Thus, in general, we would expect to use branch-and-bound search to find an optimal allocation. In this paper, we identify and solve tractable cases of OCS. In the future, we would expect to use such tractable cases to produce bounds for the general case.

We restrict the temporal constraints to be tasks ordering constraints (i.e., constraints on the relation between the start times of different tasks). For simplicity, we will continue to refer to these constraints using the same notation previously used for simple temporal constraints: $(a \leq |T_A - T_B| \leq b)$. We will also impose limits on the form $v_A(D_A)$ can take. We first assume every function $v_A(D_A)$ is a set of k_A piecewise linear functions, $\{f^k_{D_A}\}$, where $f^k_{D_A}$ is defined over an interval $[x_{k-1}, x_k]$ and $f^k_{D_A}(x_k) = f^{k+1}_{D_A}(x_k)$. Furthermore we assume each $v_A(D_A)$ is convex.

Case 1: Total Ordering with No Task Rejection

THIS CASE IS TOO LONG RELATIVE TO ITS IMPORTANCE AND TO BE SHRUNK LATER IN SUMMARY, IT IS: OneShotLP. Formulate the problem as an STPP and use the STPP2LP to find a solution via LP_Solve. Simple and fast but suitable only for FlexMinZero with NoContainment problems (rejection of a request is represented by granting it a duration of zero requests are ordered by their window start times when mapping into STPP/LP which proved to be an optimal ordering when no containment exists)

Given the above restricted version of OCS, suppose that we have predetermined the relative order of all conflicting tasks. Then, we can use the LP formulation of the STPP to solve for the task durations. The LP contains a variable representing the start, duration and end of each task (S_A, D_A, E_A). We also introduce a new variable V_A. The form of the LP is as follows: To build the LP formulation of the STPP, we have:

maximize $\sum v_A$
subject to: forall A
$S_A + D_A = E_A$
$s_{A_lb} \leq S_A \leq s_{A_{ub}}$,
$e_{A_lb} \leq E_A \leq e_{A_{ub}}$,
$s_{A_lb} \leq D_A \leq d_{A_{ub}}$,
$a \leq |T_A - T_B| \leq b$

$\forall k_A \, V_A \leq f_{D_A}^{k_A}(D_A)$

We refer the reader to (?) for the proof that the solutions to this LP are the optimal solutions to the STPP.

Extending the STPP The requirement that all conflicting tasks be ordered is not particularly natural in our case. Moreover, even then, if there are too many tasks, the system of linear constraints in the LP may be trivially infeasible. However, if it just so happens that all activities that could overlap on the resource are ordered, and all tasks allow for the possibility of zero duration then the STPP is trivially feasible, and "maximal" (no subset of the STPP has a better optimal solution).

The requirement that we be able to reduce the duration of a task to zero allows the LP algorithm to reject tasks without conducting combinatorial search. However, we must still order the tasks. We now identify a restricted set of problems for which the best possible ordering can be found in polynomial time. From the above discussion it follows that for such problems the optimization problem is polynomial (assuming zero-duration is allowed).

Definition 2 *Two requested tasks A, B are a* containment pair *if either ($s_{A_{lb}} \leq s_{B_{lb}}$ and $e_{B_{lb}} < e_{A_{lb}}$) or ($s_{A_{lb}} < s_{B_{lb}}$ and $e_{B_{lb}} \leq e_{A_{lb}}$).*

Theorem 1 *Given an OCS problem whose preferences are piecewise linear convex functions over D_A, and whose metric temporal constraints are trivially feasible and limited to time windows and task ordering. If no pair of tasks A, B is a containment pair, then the ordering according to $s_{A_{lb}}$ (and $e_{A_{ub}}$ in the case of ties) induces an STPP whose optimal solution is maximal over the STPPs induced by any other task ordering.*

Proving this theorem will be accomplished by proving an equivalent theorem introduced after the following definitions.

Definition 3 *The "natural order" of requests is the ordering induced by the start times of the associated visibility windows, with ties broken by the window end times.*

Definition 4 *In any (grounded) schedule, a two granted requests are a "mismatched pair" iff the order of their allocated times differs from the natural order.*

Definition 5 *In any (grounded) schedule, a two granted requests are "adjacent" iff there is no other request with allocated time between theirs.*

Theorem 2 *If R is a set of requested tasks with no containment pairs, R has an optimal schedule T with no mismatched pairs.*

To prove this theorem, it is enough to prove that given any schedule S with mismatched pairs, we can derive an equivalent (of the same value) schedule with no mismatched pairs. As a result, any optimal schedule can be transformed into an optimal schedule with no mismatched pairs.

First a lemma about swapping adjacent mismatched pairs, then an algorithm for deriving equivalent schedules with no mismatched pairs.

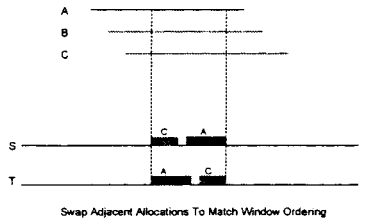

Figure 1: Swapping Adjacent Mismatched Pairs in Grounded Schedule.

Lemma 1 *Let S be any (grounded) schedule for R. Let TA and TC be any two requests, and let A and C be intervals corresponding to their allocated times in S. If A and C are adjacent and mismatched, then there is an equivalent schedule T with A and C adjacent and not mismatched.*

Proof (see illustration in figure1): Let I be the smallest interval containing the time allocated to both A and C. I is in the intersection of the visibility window of both requests satisfied by A and C, therefore the time span allocated to each of A and C may occur anywhere in I. Because A and C are adjacent, there is no other request with allocated time in I to interfere with repositioning A and C. Keeping their sizes the same (which makes the resultant schedule T equivalent to S), we may cause A and C to exchange places within I (keeping the rest of the schedule unchanged). □

Algorithm: Let S be any optimal schedule for R. Let k be the number of task allocations in S. Make $k-1$ traversals of the timeline, left-to-right: whenever two adjacent allocations appear that are mismatched, swap them as described in the lemma. After the $k-1$ traversals, there are (according the the theory of bubble sort!), no remaining mismatched pairs. Since S was optimal, so is the bubble-sorted resultant schedule. □

This completes the proof for Theorem1.

Now suppose we have a containment pair with A containing B. If we assume tasks are *interruptible* we can split A into A' and A'' such that the optimal ordering is $A' < B < A''$ according to the previous rule. Initially, this may not appear to be an equivallent problem since our preference for the duration of A'' depends on the duration of A'. However, we can use the sum of their durations directly in the objective function for the induced LP.

It is tempting to think that we can find the optimal ordering for a containment pair without interruptibility. Either ordering prunes duration choices for A but neither prune duration choices for B. It is trivial to determine which prunes

more duration choices for A (we can construct cases for which either ordering prunes more choices). One might think that the optimal ordering is that which prunes fewer duration choices for A. Unfortunately, not only is this not true, but we can demonstrate that containment pairs actually break the "optimal" ordering for non-containment pairs of tasks. The following simple example (also shown in Figure **??**) shows this:

Task A: $s_{A_{lb}} = 0, e_{A_{ub}} = 5, d_{A_{ub}} = 2, v_A(D_A) = 2d_A$
Task B: $s_{B_{lb}} = 1, e_{B_{ub}} = 6, d_{B_{ub}} = 2, v_B(D_B) = d_B$
Task C: $s_{C_{lb}} = 0, e_{C_{ub}} = 1, d_{C_{ub}} = 1, v_C(D_C) = 10d_C$
Task D: $s_{D_{lb}} = 2, e_{D_{ub}} = 3, d_{D_{ub}} = 1, v_D(D_D) = 10d_D$
Task E: $s_{E_{lb}} = 5, e_{E_{ub}} = 6, d_{E_{ub}} = 1, v_E(D_E) = 10d_E$

The idea is that highest paying tasks (C, D,and E) should be placed in the schedule and given their maximum duration. This leaves 2 gaps to be filled by lower paying tasks (A and B). The gaps are as follows: [1 2] and [3 5]. If $A < B$, A gets assigned [1 2] and B gets assigned [3 5]. The value is 2+2=4. If $B < A$, B gets assigned 1 slot [1 2] and A gets assigned 2 slots [3 5]. The value is 1+4=5 Thus, $B < A$ is the optimal ordering for A and B, contradicting the optimal ordering when no containment pairs are present.

Case 2: Fixed Duration with Task Rejection

Overlap Boundness implies apolynomial run time (exponential only in the 'bound').

Simple Solver For each time point on the horizon, consider all possible allocations. Pruning is done similarly to the next algorithm.

PrefixLengthBased Partition Solver Based on a conceptual tree. Nodes at level i represents the i^{th} request in the partial schedule starting from the node down the branch to this node. At each node, the requests are split into 2 sets: 'closed' (past their last opportunity) and 'open' (still active). For each node, the first child is the request that ends the earliest among the 'open' and un-sused requests. The remaining children are all 'open' and un-used requests that 'may' start any time between the end of the node and the end of the first child. Pruning of a partial schedule is based on the combination of three parameters:

TimeBased Partition Solver Partition the time horizon based on 'latest start time' of the requests. For each partition (bucket), consider all possible extensions to all partial schedules received from previous partition. Prune similarly to 3a. Adjust the 'closed', 'active', and 'open' sets. A request is 'closed' if its past its last opportunity, 'active' if it could start within the current partition (bucket), and 'open' otherwise.

Notes on the three solvers for Fixed Durations The Simple and PrefixLengthBased Partion are implemented with similar domination rules but the later seems to outperform the former for problems that have many competing requests. The PrefixLengthBased Partion has a wider (fatter) but shallower (shorter) search tree than the .

The domination rules of the PrefixLengthBased Partition are applied to subschedules that are of the same size in terms of number of grants while the domination rules of the Simple are applied to subschedules that are of the, roughly, same extension.

The dominations rules of PrefixLengthBased could be enhanced by taking into consideration when the difference in component sets has a value that doesn't exceed the difference in value of 2 subschedules, comparing a subschedule with prefixes of other subschedules to allow for 'fair' comparisons in terms of extensions, and comparing a subschedule with dummy zeros added, when needed before attaching an actual grant, with one-level prefix (one resulting from ignoring the last grant only) of other subschedules to allow for elimination prior to adding remaining siblings

Case 3: Flexible Duration with Task Rejection

If the flexible duration is chosen in a discrete manner, then the solvers for fixed duration may be extended to handle all the discrete possibilities of the flexible durations. This adds to the complexity a factor of the discrete flexibility in the duration. A more interesting case is when the flexible duration is over continuous domain and, therefore, this what we will explore next.

Dual Graph Based Solver Build the Dual graph where the nodes are each pair of requests with overlapping windows. The domain for each node is a set of 5 values indicating if any of the component requests is to be granted and, if both granted, in which order. The values selected for the nodes under any single scenario has to be consistent. (obvious?) For each scenario, which represents a selection of subtasks to grant in some specific ordering, an LP problem is formed and LP_Solve is called to find the optimal allocation. Filtering is done prior to forming the LP problem if the scenario is found to be no better than the best currently found scenario. Semi-simple and slow but suitable for least restricted problems. Dealing with OvrlpBounded problems implies having a bounded induced width graphs which results in complexity reduction. (how much reduction?)

Partition Based Solver Divide the time horizon into partitions based on 'earliest start time' of the requests. For each partition (bucket), let 'fresh' be the set of requests that initiated this partition (ones that their earliest start time is at the start of this partition). 'closed' is the set of requests whose latest start time precedes this partition. 'active' is the set of requests that could start at this partition. For each partial schedule from the previous partition, let 'flow' be the set of requests in this schedule that ends beyond the start of current partition. For each partial schedule from the previous partition, for each possible selection of a subset from 'fresh', for each possible insertion of this subset in 'flow', form an LP problem (selected requests and ordering), include maximum left shift of grants in the objective function (the end time of the last request is multiplied by epsilon, the inverse of the minimum bid (?), and added to the objective function), extract needed information from LP solution. Prune accordingly using the concept of Independent Prefix (more on this soon). This solver is Complex and Slow but suit-

generateProblem(h,w,d,v,p,n)

for n tasks
  Choose window size w_n u.a.r. from [1..w]
  Choose start time u.a.r. from $[1..h - w_n]$
  while task window *contains* another task window
    Shrink window; update w_n
  while task window *contained by* another task window
    Enlarge window; update w_n
  Choose duration d_n u.a.r. from $[1..\min(d, w_n)]$
  Choose maximum bid b_n u.a.r. from $[1..v]$
  Choose number of pieces of preference u.a.r. from $[1..p]$
  for each piece of preference
    $\#d*$ is remaining duration w_n, $b*$ is remaining bid height b_n
    Choose piece duration d_i u.a.r. from $[1..d*]$; update d*
    Choose bid height b_i u.a.r. from $[1..b*]$; update b*
    Add preference function piece of duration d_i up to bid b_i

able for complex problems where durations are flexible with continuous value assignments.

Empirical Results

In this section we describe some preliminary empirical results on the Case 1 simplification applied to randomly generated instances of these problems.

Random Problems

The instances are generated as follows. Let h be the scheduling horizon. Let w be the maximum window for any task, d be the maximum duration of any task, v be the maximum bid for any task, and p be the maximum number of pieces of the preference function, also inputs. First, the task start time and maximum duration are selected uniformly from the given range. These characteristics are then modified to ensure that no pair of tasks is a containment pair. Finally, the maximum bid of the task is generated, a random number of pieces of the preference function is selected, and the preference function is generated.

Preliminary Empirical Results

We present the results of preliminary empirical tests designed to show the growth in time to solve the LP as a function of the problem [1]. Due to the relative simplicity of the temporal constraints on tasks, the principal contributors to the LP are the number of tasks and the number of pieces in tasks' preference functions. Figure **??** shows timing results for problems with $h = 4000, w = 20, d = 15$ and $v = 40$. We vary the number of tasks n from 200 to 1600 by 200, and the maximum number of pieces of the bid p from 1 to 6. Growth in problem complexity is sub-linear in p, as expected, and appears polynomial in the number of tasks. The CPU times shown are in seconds; we average over 5 randomly generated problems. Timing is only for solving the LP, which was done by lpsolve, a public domain LP solver. More results could be added from CASE 2

[1] Investigations of solution quality, task rejection and so on are pending.

Conclusions and Future Work

We describe the OCS problem for Deep Space Network. This problem consists of activities with time windows and flexible durations, simple temporal constraints, and preferences over a combination of start time and duration of tasks. We have shown that a VCG mechanism exists for this problem, which (to our knowledge) is the first mechanism for an auction on a mixture of divisible and indivisible goods. We have identified a class of OCS problems with a tractable second-price mechanism: activities with time windows, flexible duration, ordering constraints, convex, piecewise linear preferences, no containment pairs, and for which duration of zero is feasible. The tractability result is a novel extension of the theory of STPPs to tractably handle task ordering and task rejection.

Obviously a study of the general case of OCS for the DSN is worthwhile. Task containment, metric temporal constraints, multi-capacity resources and choices over resources will generally lead to $\mathcal{NP}$-hard problems. We are investigating options for incorporating the solving of tractable subcases in a complete solver, in a manner similar to that described in (**?**). The tractable cases for STPP described in this paper provide methods for bounding above the value of schedules by relaxing the STPP constraints; they can then be used both as inference mechanisms and as the bases of heuristics. Proving the value of the bounds and subsequently experimenting with those bounds inside a complete solver are on our agenda.

In our formulation, we implicitly assumed that the facility owner (NASA) was the auctioneer and thus had no bid. However, we could generally introduce new bidders to represent the facility owner's criteria on schedules. For example, maximizing utilization could be a preference; NASA could be given a budget for this, and express preferences over it, thereby influencing the auction. In general, we believe the tractability results will extend to multiple preferences over a set of activities.

The tractable cases of OCS for DSN restrict the nature of the preference functions to piecewise linear convex functions of the duration, $v_A(D_A)$ and forces preferences to extend to $D_A = 0$. If more general preference functions $v_A(D_A, S_A)$ are allowed, care must be taken to ensure that rejected tasks derive no value, or formally, $v_A(0, S_A) = 0$. Furthermore, metric temporal constraints on rejected tasks should not be enforced. The STPP framework may not be appropriate in this setting; we are investigating other formalisms in search of new tractable algorithms.